

REMARKS

I. Introduction

In response to the Office Action dated March 11, 2004, no claims have been cancelled, amended or added. Claims 1-18 remain in the application. Re-examination and re-consideration of the application, as amended, is requested.

II. Specification Objections

In paragraph (2) of the Office Action, the Abstract of the Disclosure was objected to as exceeding 150 words in length.

Applicants' attorney submits herewith a replacement Abstract of the Disclosure to overcome this objection.

III. Double-Patenting Rejection

In paragraphs (3)-(4) of the Office Action, claims 1-18 were rejected under the judicially created doctrine of obviousness-type double patenting as being respectively unpatentable over claims 1-5 of U.S. Patent No. 6,175,957 B1.

Applicants' attorney submits herewith a Terminal Disclaimer under 37 C.F.R. §1.321(c) to overcome these rejections.

IV. Prior Art Rejections

A. The Office Action Rejections

In paragraphs (5)-(6) of the Office Action, claims 1, 4, 7, 10, 13, and 16 were rejected under 35 U.S.C. §103(a) as being unpatentable over Pettis et al., "Profile Guided Code Positioning," ACM 1990 (Pettis) in view of Tomko et al., "Profile Driven Weight Decomposition," ACM 1996 (Tomko). However, in paragraphs (7)-(8) of the Office Action, 2-3, 5-6, 8-9, 11-12, 14-15 and 17-18 were objected as being dependent upon a rejected base claim, but would be allowable if rewritten in independent form to include the base claim and any intervening claims, provided that a Terminal Disclaimer was submitted to overcome the double-patenting rejection.

Applicants' attorney acknowledges the indication of allowable claims, but respectfully traverse these rejections.

B. Applicants' Independent Claims

Independent claims 1, 7, and 13 are generally directed to a method of restructuring a program comprising basic blocks for execution by a processor having a memory hierarchy comprised of a plurality of levels. A Program Execution Graph (PEG) corresponding to a first level of the memory hierarchy is constructed from control flow and frequency information from a profile of the program. The PEG comprises a weighted undirected graph comprising nodes and edges, wherein each node represents a basic block, each edge represents a transfer of control between a pair of the basic blocks, each of the nodes has a weight equal to the size of the basic block represented by the node, and each of the edges has a weight equal to a frequency of transition between a pair of basic blocks represented by a pair of nodes connected by the edge. For the first level of the memory hierarchy, the nodes of the PEG are partitioned into clusters, such that a sum of weights of the edges whose endpoints are in different clusters is minimized, and such that, for any cluster, a sum of weights of the nodes in the cluster is no greater than an upper bound corresponding to a size of the first level of the memory hierarchy. The basic blocks are restructured into contiguous code corresponding to the clusters, such that the basic blocks that communicate extensively with each other are on a same level of the memory hierarchy, in order to reduce communications between the basic blocks across the levels of the memory hierarchy.

C. The Pettis Reference

Pettis describes profile-guided code positioning wherein execution profile data is used as input into the compilation process. One prototype positions code based on whole procedures, wherein it has the ability to move procedures into an order that is determined by a "closest is best" strategy. Another prototype positions code based on basic blocks within procedures, wherein basic blocks that would be better as straight line sequences are identified as chains and these chains are then ordered according to branch heuristics.

D. The Tomko Reference

Tomko describes profile driven weighted decomposition, wherein application and machine specific information are used in conjunction with domain decomposition to achieve a level of performance not possible with traditional domain decomposition methods. Application profiling characterizes the performance of an application on a specific machine. A method is presented that

uses curve-fitting of application profile data to calculate vertex and edge weights for use with weighted graph decomposition algorithms.

E. The Applicants' Claimed Invention Is Patentable Over The References

Applicants' independent claims are patentable over the references because they recite a novel and nonobvious combination of elements. Nonetheless, the Office Action states the following:

As per claim 1:

Pettis discloses a method for constructing an undirected weighted graph. The constructing graph is partitioned into sub-graph due to the edge weights and the size of a procedure (basic block). The teaching cover the claim limitation hereafter:

a) constructing a Program Execution Graph (PEG) corresponding to a first level of the memory hierarchy (Page 16, right column, lines 27-32) from control flow and frequency information from a profile of the program, the PEG comprising a weighted undirected graph comprising nodes and edges

each node representing a basic block, each edge representing a transfer of control between a pair of the basic blocks (Page 17, right column, see section 3.2, 'each node of the graph is a single procedure and the edges correspond to calls between procedure'. 'A single procedure' is also referred as a basic block - page 21, right column, section 5), each of the nodes having a weight equal to the size of the basic block represented by the node and

each of the edges having a weight equal to a frequency of transition between a pair of basic blocks (See page 17, right column, section 3.2, 'the edges correspond to calls between procedure') represented by a pair of nodes connected by the edge;

b) for the first level of the memory hierarchy, partitioning the nodes of the PEG into clusters, such that a sum of weights of the edges whose endpoints are in different clusters is minimized (See page 21, left column fourth paragraph, referring to the partition: A, E-N-B-C-D-F-H, I-J-L, G-O, K, M' partitioned from the graph of Figure 4 in page 20), and such that for any cluster, a sum of weights of the nodes in the cluster is no greater than an upper bound corresponding to a size of the first level of the memory hierarchy,

c) restructuring the basic blocks into contiguous code corresponding to the clusters, such that the basic blocks that communicate extensively with each other are on a same level of the memory hierarchy, in order to reduce communications between the basic blocks across the levels of the memory hierarchy (Page 21, see entirely section 4.3, Basic Block Placement, 'the final phase is to restructure the basic block graph before passing to the actual optimizer').

Pettis does not explicitly include the "node weights"; however, Pettis does teach about basic block splitting for minimizing the size of the basic block; thus it balances pages and accounts for cache misses (Page 21, right column, section 5. Procedure Splitting, see second paragraph, referring to 'Procedure Splitting is process of separating the fluff basis blocks of a procedure into a separate region in attempt to minimize the size of the primary procedure').

Tomko discloses a graph that is included with vertex weights (node weights) and edge weights for use with graph partitioning (Re Tomko: page 165, right

column, first paragraph). It discloses a method for partitioning the vertices (nodes) into the number of disjoint subset so that the sum of the vertex weights for the subset with the highest sum is close to the average sum, and the total cost of the cut edge is minimized (Re Tomko: page 166, left column, section 3, Domain Decomposition, third paragraph, "The weighted graph partitioning ...").

Therefore, it would have been obvious to a person of ordinary skill in the art at the time of invention was made to combine the teaching of Pettis: Partitioning weighted graph where the weights are in the edges, with the teaching of Tomko: Partitioning the weighted graph where the weights are with the vertices and edges. Doing so would yield balance loading and conform to the size limitation of certain memory/register resources, thus would improve the performance of compilations for profiling.

Applicants' attorney disagrees. At the indicated locations, the combination of Pettis and Tomko does not teach or suggest all the limitations of Applicants' claims.

For example, the Office Action cites Pettis as teaching the limitations of Applicants' original independent claims directed to "b) for the first level of the memory hierarchy, partitioning the nodes of the PEG into clusters, such that a sum of weights of the edges whose endpoints are in different clusters is minimized, and such that for any cluster, a sum of weights of the nodes in the cluster is no greater than an upper bound corresponding to a size of the first level of the memory hierarchy," at page 21, left column fourth paragraph.

However, the cited location in Pettis merely describes the following:

Pettis: page 21, left column fourth paragraph

After we determine the precedence criteria between chains, we arrange them so that the order we want is satisfied as best as possible. If there is some freedom in choosing the next chain, we try to choose the chain connected to existing chains by the heaviest count. In this example, the final order of the basic blocks that we determine is:

A, E-N-B-C-D-F-H, I-J-L, G-O, K, M

Note that this ordering has altered the layout of the loop. In particular, the main test for the loop (basic block B) is now in the middle of the loop.

This portion of Pettis merely describes choosing chains connected to existing chains by the heaviest count. However, nowhere does it describe partitioning the nodes of the PEG into clusters, such that a sum of weights of the edges whose endpoints are in different clusters is minimized, and such that for any cluster, a sum of weights of the nodes in the cluster is no greater than an upper bound corresponding to a size of the first level of the memory hierarchy, as recited in Applicants' claimed invention.

The Office Action also cites Pettis as teaching the limitations of Applicants' original independent claims directed to "c) restructuring the basic blocks into contiguous code corresponding to the clusters, such that the basic blocks that communicate extensively with each other are on a same level of the memory hierarchy, in order to reduce communications between the basic blocks across the levels of the memory hierarchy," at page 21, section 4.3.

However, the cited locations in Pettis merely describe the following:

Pettis: page 21, section 4.3
4.3. Basic Block Placement

After determining the set of chains and their ordering, the final phase is to restructure the basic block graph before passing into the actual optimizer. Because internally the basic block graph is built using a doubly linked list of instructions, what really must be done is to properly break and reconnect the instructions at the basic block boundaries. By and large, this process just involves pushing pointers around.

We start by placing the chains down, reconnecting the basic blocks to each other by means of the double links. Then, a pass is made over the basic blocks to insert any necessary branches to ensure that the code executes correctly. These branches are inserted where the fall-through after a basic block is no longer correct. For example, if the then clause of an if-then-else statement was moved away, the else basic block would immediately follow the branch and it would be in the fall through location; however, the correct fall-through for the branch is the then clause. For correctness, a new basic block with a branch to the original fall-through location is inserted to be the fall-through.

This means that the conditional branch in the if clause is a branch around an unconditional branch. We could take the trouble to reverse the sense of the conditional branch and alter the target rather than just inserting a new basic block, but the optimizer already knows how to do this so we let it do the work later.

At this point, the procedure is ready to be passed through the optimizer.

These portions of Pettis merely describe restructuring the basic block graph and reconnecting basic blocks, including the inserting of any necessary branches to ensure that the code executes correctly. However, nowhere does it describe restructuring the basic blocks into contiguous code corresponding to the clusters, such that the basic blocks that communicate extensively with each other are on a same level of the memory hierarchy, in order to reduce communications between the basic blocks across the levels of the memory hierarchy, as recited in Applicants' claimed invention.

Finally, the Office Action admits that Pettis does not teach "node weights," but asserts that Pettis does teach basic block splitting for minimizing the size of the basic block at page 21, right column, section 5, second paragraph, and that Tomko discloses a graph that is included with vertex weights (node weights) and edge weights for use with graph partitioning, and that Tomko discloses a

method for partitioning the vertices (nodes) into the number of disjoint subset so that the sum of the vertex weights for the subset with the highest sum is close to the average sum, and the total cost of the cut edge is minimized, at page 166, left column, section 3, third paragraph. According to the Office Action, it would have been obvious to a person of ordinary skill in the art at the time of invention was made to combine the teaching of Pettis, e.g., partitioning weighted graph where the weights are in the edges, with the teaching of Tomko, e.g., partitioning the weighted graph where the weights are with the vertices and edges, because doing so would yield balance loading and conform to the size limitation of certain memory/register resources, thus would improve the performance of compilations for profiling.

The cited locations in Pettis and Tomko are set forth below:

Pettis: page 21, right column, section 5, second paragraph

Procedure Splitting is the process of separating the fluff basic blocks of a procedure into a separate region in an attempt to minimize the size of the primary procedure. The benefits of procedure splitting magnify the concept of locality. By producing smaller and denser primary procedures, more procedures can now be packed onto a single page. This should result in a further reduction of the page working set size and the number of page and TLB misses:

Tomko: page 165, right column, first paragraph

Applications with irregular domains are a challenge to parallelize efficiently, and heuristic algorithms are used to partition them for parallel execution. Many such applications are even more challenging due to the heterogeneity of their domains: partitioning the application into equal size sub-domains does not guarantee load balance because the work load per vertex varies across the application domain. Fortunately, several domain decomposition algorithms are available which allow weighted graphs as input. These algorithms can potentially deal with such heterogeneous applications. However, these domain decomposition algorithms are only half of the solution; without an appropriately weighted in-put graph they can not do an adequate job. We present a method that uses curve-fitting of application profile data to calculate vertex and edge weights for use with weighted graph partitioning algorithms. Our method is the first of which we are aware to solve the problem of graph weight calculation. We demonstrate its potential on some important routines from a production finite element application. Our method combined with a weighted multilevel algorithm reduced load imbalance from 52% to less than 10% on the more imbalanced subroutine of the two in our case study.

Tomko: page 166, left column, section 3, third paragraph

The weighted graph partitioning problem can thus be stated as follows. Given a graph with vertex and edge weights, partition the vertices into some number of disjoint subsets such that sum of the vertex weights for the subset with the highest sum is close to the average sum, and the of the cut edges is minimized.

These portions of Pettis merely describe procedure splitting, while Tomko merely describes a method that uses curve-fitting of application profile data to calculate vertex and edge weights for use with weighted graph partitioning algorithms and the weighted graph partitioning itself. However, nowhere do they describe: (1) partitioning the nodes of the PEG into clusters, such that a sum of weights of the edges whose endpoints are in different clusters is minimized, and such that for any cluster, a sum of weights of the nodes in the cluster is no greater than an upper bound corresponding to a size of the first level of the memory hierarchy, as recited in Applicants' claimed invention; or (2) restructuring the basic blocks into contiguous code corresponding to the clusters, such that the basic blocks that communicate extensively with each other are on a same level of the memory hierarchy, in order to reduce communications between the basic blocks across the levels of the memory hierarchy, as recited in Applicants' claimed invention. Moreover, these limitations are not be obvious in view of the combination of Pettis and Tomko.

Thus, Applicants' attorney submits that independent claims 1, 7 and 13 are allowable over Pettis and Tomko. Further, dependent claims 4, 7, 10 and 16 are submitted to be allowable over Pettis and Tomko in the same manner, because they are dependent on independent claims 1, 7 and 13, respectively, and thus contain all the limitations of the independent claims. In addition, dependent claims 4, 7, 10 and 16 recite additional novel elements not shown by Pettis and Tomko.

V. Conclusion

In view of the above, it is submitted that this application is now in good order for allowance and such allowance is respectfully solicited.

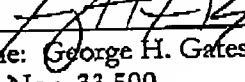
Should the Examiner believe minor matters still remain that can be resolved in a telephone interview, the Examiner is urged to call Applicants' undersigned attorney.

Respectfully submitted,

GATES & COOPER LLP
Attorneys for Applicants

Howard Hughes Center
6701 Center Drive West, Suite 1050
Los Angeles, California 90045
(310) 641-8797

Date: June 9, 2004

By: 
Name: George H. Gates
Reg. No.: 33,500

GHG/